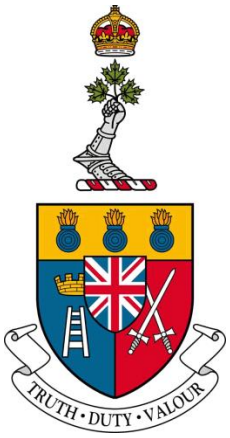
Royal Military College of Canada

Department of Electrical and Computer Engineering



Detailed Design Document

(EEE455/457-DID-07)

for

Targeted Encryption Device

NCdt Blake Mackey

OCdt Jason Leverton

Project EEE/GEF 455/457-15-13

Supervisor: Dr Sylvain Leblanc

Project Manager: Maj Randy Hartman

1 April 2015

Table of Contents

Table of Contents ii
List of Figures and Tables v
List of Abbreviations vii
1 Introduction 1
1.1 Document Purpose 1
1.2 Background 1
1.2.1 Proposed Use Cases 1
1.2.1.1 Secure Endpoint 2 to Secure Endpoint 4 2
1.2.1.2 Secure Endpoint 2 to Non-Secure Endpoint 3 2
1.2.1.3 Non-Secure Endpoint 1 to Secure Endpoint 4 2
1.2.1.4 Non-Secure Endpoint 1 to Non-Secure Endpoint 3 2
1.3 Aim 2
1.4 Scope 2
1.4.1 TED Capabilities 3
2 Applicable Documents 4
3 Product Requirements 6
3.1 Functional Requirements 6
3.2 Interface Requirements 6
3.3 Performance/Timing Requirements 6
3.4 Implementation Restrictions 6
4 TED Architecture 7
5 Detailed Design 10
5.1 Overview 10
5.2 Limitations 10
5.2.1 NAT Operation 10
5.3 Modules 10
5.3.1 Cryptographic Module 10
5.3.1.1 Cryptographic Design Specification 10
5.3.1.2 Cryptographic Design 10

5.3.1.3 Internal Structure....12
5.3.1.4 Operation....12
5.3.1.5 Error Handling....14
5.3.2 CPU/Memory Module....14
5.3.2.1 CPU/Memory Design Specification....14
5.3.2.2 CPU/Memory Design....14
5.3.2.3 Internal Structure....16
5.3.2.4 Operation....18
5.3.2.5 Error Handling....19
5.3.2.6 Miscellaneous....19
5.3.3 DMA Module....20
5.3.3.1 DMA Design Specification....20
5.3.3.2 DMA Design....20
5.3.3.3 Internal Structure....21
5.3.3.4 Operation....22
5.3.3.5 Error Handling....23
5.3.4 Parallel I/O Module....23
5.3.4.1 Parallel I/O Design Specification....23
5.3.4.2 Parallel I/O Design....24
5.3.4.3 Internal Structure....24
5.4 Interface Descriptions....24
5.4.1 All FPGA Internal Interfaces....25
5.4.2 Ethernet....25
5.4.3 LCD....25
5.4.4 Parallel I/O....26
5.4.5 SDRAM....27
6 Equipment....28
7 Results....29
7.1 Testing....29
7.1.1 Bandwidth Test....29

7.1.2 Latency Test 29
7.1.3 Duplexity Test 30
7.1.4 NAT Test 30
7.1.5 Administration Panel Test 30
8 Summary 32
9 Conclusion 33
10 Discussion 34
10.1 PR-04 Requirement Removal 34
10.2 Major Challenge and Lessons Learned 34
11 Future Work 35
11.1 Cryptographic Cipher Iterations 35
11.2 Packet Processing Depth 35
Annexes 36
Annex A – Additional Diagrams and Figures 36

List of Figures and Tables

Figure 1-1: TED Implementation 1
Figure 4-1: Architecture Model 7
Figure 4-2: Altera Hardware/Software Tool Suite 8
Figure 4-3: TED Development Flow 9
Figure 5-1: Demonstration of Weak Cryptographic Hash 11
Figure 5-2: Encryption Structure for a Block Chain Cipher 11
Figure 5-3: Decryption Structure for a Block Chain Cipher 11
Table 5-1: Cipher Output with no Salt or Rotation 12
Table 5-2: Cipher Output with Salt 13
Table 5-3: Cipher Output with Rotation and Salt 13
Formula 5-1: Block Repetition 14
Figure 5-4: Ethernet and IPv4 Headers 15
Figure 5-5: TCP Header 15
Figure 5-6: UDP Header 15
Figure 5-7: Packet Processor Initial View 16
Figure 5-8: Packet Parser Final View 16
Figure 5-9: Packet Processor Structure 17
Figure 5-10: Packet Processor Data Structure 18
Figure 5-11: Packet Processor Flow Diagram 19
Figure 5-12: Admin Packet Data Structure 20
Figure 5-13: DMA members 21
Figure 5-14: Typical DMA Transfer of Control (Incoming) 21
Figure 5-15: Typical DMA Transfer of Control (Outgoing) 22
Figure 5-16: Packet Buffer Linked Lists 22
Figure 5-17: Terasic DE2-115 Development Board Interfaces 25
Figure 5-18: Ethernet Interfaces 25
Figure 5-19: LCD Interface 26
Figure 5-20: DE-115 Switches Interface 26
Figure 5-21: DE-115 LEDs Interface 27

Figure 5-22: DE-115 Switches Interface 27
Figure 5-23: SDRAM Interface 27
Table 6-1: Final Project Equipment List 28
Figure 7-1: Comparison of Bandwidth with and without TED 29
Figure 7-2: Latency of Variable Packet Size 30
Figure 7-3: TED Administration GUI 31
Figure 7-4: TED UDP Administration Packet 31
Figure A-1: Block Cipher Decryption Model 36
Figure A-2: Block Cipher Encryption Model 37

List of Abbreviations

CPU	central processing unit
DMA	direct memory access
EOP	end of packet
FPGA	field programmable gate array
GUI	graphical user interface
HDL	hardware description language
IV	initialization vector
LAN	local area network
LED	light emitting diode
LCD	liquid crystal display
MAC	media access controller
MII	media independent interface
MM	memory-mapped
NAT	network address translation
PHY	physical layer interface (chip)
RAM	random-access memory
ROM	read-only memory
SOP	start of packet
SBT	software build tools (Altera Inc. proprietary Eclipse distribution)
SDRAM	synchronous dynamic random-access memory
TED	targeted encryption device
VHDL	very high speed integrated circuit hardware description language
VPN	virtual private network

1 Introduction.

1.1 Document purpose.

This document will describe the design of our project the Targeted Encryption Device (TED). It will describe what engineering problems were faced, how we approached these problems, and how we solved these problems. This paper will also define the requirements of this device and present the testing to support that the requirements were met.

1.2 Background.

A Targeted Encryption Device is a hardware implementation that will secure network data communication between itself and another TED, which are used as a pair between two secure endpoints. Figure 1-1 shows how the TED will operate inside a simple network. The TED will not be addressable on the network, and will be transparent in the network communication process.

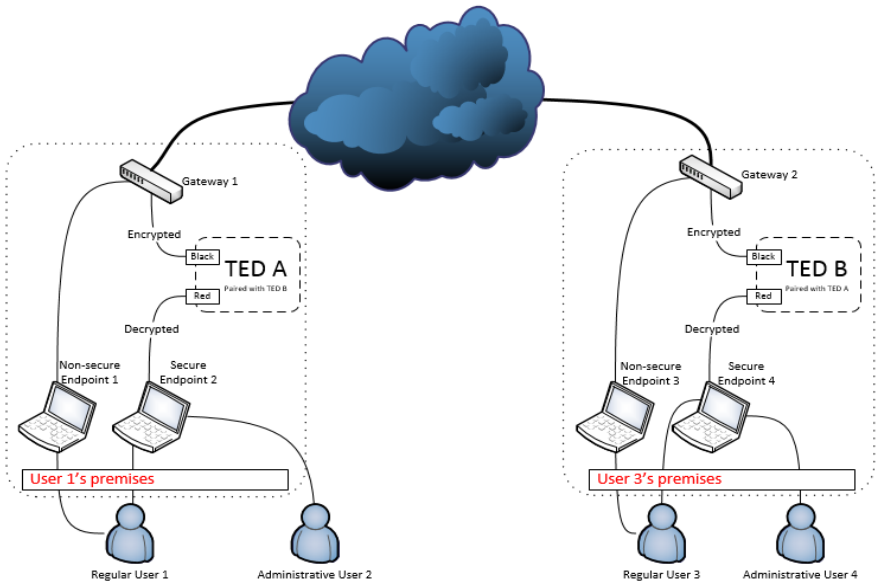


Figure 1-1: TED Implementation

Virtual private networks (VPN) are currently the most common solution to secure communications between two endpoints. Often these endpoints require the same protocol, configuration and encryption to speak to each other. VPNs that run on an operating system have an inherit risk that some information intended to be secured, was not.

There are a number of flaws as a result of this implementation. The major flaws that we intend to address are the following: memory leaks from software running on the secure endpoint; detection of the VPN signature by packet signature categorization; and/or an accidental plaintext transmission. The TED will overcome these limitations and vulnerabilities associated with VPN software by abstracting the encryption and configuration from secure endpoints to our proposed hardware solution.

1.2.1 Proposed Use Cases.

The four proposed use cases will be described in reference to Figure 1-1.

1.2.1.1 Secure Endpoint 2 to Secure Endpoint 4.

The main use case describes two separate *secure endpoints*, (secure endpoint 2, and secure endpoint 4) whose network traffic will be encrypted by two TEDs. All network traffic originating from *secure endpoint 2* will be received by *TED A* through the *red interface*. TED A will then inspect the details of each frame of data and then choose to encrypt the layer 3 payload and retransmit the frame. The destination is *secure endpoint 4*, the layer 3 payload is then encrypted and a new layer 4 header is generated and inserted into the frame along with the encrypted payload. This new frame is retransmitted through the black interface of TED A to *gateway 1* and routed to *secure endpoint 4*. When the frame is received by TED B through the black interface, it is inspected to determine if the source IP address matches *secure endpoint 2* and that the destination matches *secure endpoint 4*. If both of these conditions are met, the layer 3 payload is decrypted and the original frame as transmitted by *secure endpoint 2* is retransmitted on the red interface to *secure endpoint 4*.

1.2.1.2 Secure Endpoint 2 to Non-Secure Endpoint 3.

Network Traffic generated by *secure endpoint 2* is received by *TED A* through the *red interface*. The frame is actively ignored because it is not destined for a secure endpoint and therefore is transmitted on the *black interface* of *TED A* to *gateway 1*. The frame is routed to *non-secure endpoint 3* as any normal frame.

1.2.1.3 Non-Secure Endpoint 1 to Secure Endpoint 4.

Network traffic generated by *endpoint 1* is received by gateway1 and routed to *secure endpoint 4*. The packet is received by the *black interface* of *TED B*. The frame is inspected to see if the source address is originating from a secure endpoint, is actively ignored by *TED B*, and retransmitted through the *red interface* to *secure endpoint 4*.

1.2.1.4 Non-Secure Endpoint 1 to Non-Secure Endpoint 3.

This is a standard network communication.

1.3 Aim.

The aim of our project is to design and prove the concept of a pair of TEDs, capable of operating in line between a secure endpoint and a gateway to provide a secure method of network communication.

The intent is to abstract the encryption and configuration from *secure endpoints*. In order to provide transparent operation, our solution must be able to process the data coming from a secure endpoint, determine if it is destined for its pair, and whether to encrypt or actively ignore, and then retransmit the frame through the black interface to the gateway, and the inverse is true of decryption.

1.4 Scope.

1.4.1 TED Capabilities.

The TED encrypts network traffic with a custom cryptographic module. A custom cryptographic module presents a lower attack vector than an open source cipher due to its less-commonly used nature, and the non-readily available tools and scripts for decryption. The downside of a custom designed cryptographic module is the lack of any formal peer review or validation process. The system design is modular so alternate block ciphers can be used in place of the custom cryptographic module.

The TED operates transparently on any Ethernet switched local area network (LAN). In line operation between a secure endpoint and gateway, ensures all network traffic generated by or destined for the secure endpoint will be intercepted by the applicable *red* or *black* interface of the TED. This eliminates the need for addressable interfaces.

The TED operates across network address translation (NAT) configured networks. The knowledge of the public IP addresses for each host is required.

2 Applicable Documents.

[1] DE2-115 User Manual 1 Jan. 2010. Altera Inc. Web. 5 Nov. 2014. <ftp://ftp.altera.com/up/pub/Altera_Material/12.1/Boards/DE2-115/DE2_115_User_Manual.pdf>.

This user manual was used in order to configure and use the FPGA development board for the TED, the Terasic DE2-115.

[2] IEEE Working Group. "IEEE 802.3 ETHERNET." *IEEE 802.3 ETHERNET*. IEEE, 23 Aug. 2014. Web. 7 Sept. 2014. <http://www.ieee802.org/3/>.

The 802.3 standard was used to deconstruct Ethernet frames into the different layers, as well to implement full-duplex and half-duplex flow control into the TED red and black interfaces.

[3] IETF. "RFC 793 - Transmission Control Protocol." *RFC 793 - Transmission Control Protocol*. IETF, 1 Sept. 1981. Web. 7 Sept. 2014. <http://tools.ietf.org/html/rfc793>.

RFC 793 was used specifically to determine the layer four header structure as it relates to TCP in order to perform deep packet inspection of TCP IP packets.

[4] IETF. "RFC 791 - Internet Protocol." *RFC 791 - Internet Protocol*. IETF, 1 Sept. 1981. Web. 7 Sept. 2014. <http://tools.ietf.org/html/rfc791>.

RFC 791 was used in order to determine the layer three header structure as it relates to IP in order to perform deep packet inspection of IP packets.

[5] Nios II Software Developer's Handbook 13 June 2013. Web. 9 Nov. 2014. <www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf>.

The NIOS II software developers' handbook was used when writing C code for the NIOS II processor in order to conform to best practices when programming this particular processor.

[6] Nios II Hardware Development Tutorial 11 May 2011. Altera Inc. Web. 9 Nov. 2014. <www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf>.

The NIOS II hardware developers' tutorial was used when developing the custom cryptography module for the QSYS system builder in order to be able to integrate the module with the Avalon bus, and in order to write hardware drivers to integrate the module with the NIOS II processor architecture.

[7] Postel, J.. "RFC 768 - User Datagram Protocol." *RFC 768 - User Datagram Protocol*. IETF, 28 Aug. 1980. Web. 7 Sept. 2014. <http://tools.ietf.org/html/rfc768>.

RFC 768 was used in order to determine the layer four header structure as it relates to UDP in order to perform deep packet inspection of UDP IP packets.

[8] "RMC 4th Year Project Page." *Segfaults.net*. Project Management Office EEE455/457, 1 Sept. 2013. Web. 7 Sept. 2014. <http://projects.segfaults.net/joomla/index.php/en/statement-of-work>.

The RMC 4th year project page was used as a reference for timelines and document templates in order to conform with the 4th year final project standards and deadlines.

[9] Quartus II Handbook 18 Aug. 2014. Altera Inc. Web. 5 Nov. 2014. <http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf>

The Quartus II handbook was used as a general reference when implementing all parts of the TED in order to provide a reference when using different programs in the suite (QSYS, Quartus II, NIOS SBT).

3 Product Requirements.

3.1 Functional Requirements.

FR-01: *Network Visibility* – The device must be in-line of data transmission in order to achieve visibility and control over all packets transmitted and received by the *secure endpoint*.

FR-02: *TED Pairing* – TEDs will work in units of two. Both TEDs will have a shared secret key and operate point to point.

3.2 Interface Requirements.

IR-01: *Network Access* – TED will be connected directly to the *secure endpoint* and the *gateway* with RJ45.

IR-02: *NAT Configuration* – Static NAT setup on the *gateway* will be required if NAT mode is enabled.

IR-03: *Admin User Configuration* – Develop a graphical user interface capable of sending configuration commands to the TED from a *secure endpoint*.

3.3 Performance/Timing Requirements.

PR-01: *Device Network Speed* - The TED must be able to operate in real-time with the *secure endpoint* and the *gateway* at 100Mbps.

PR-02: *TED Processing* - The delay of processing the encryption decision will not exceed 10mS.

PR-03: *Duplexity* – The TED must be able to send and receive concurrently on its *red* and *black* ports.

3.4 Implementation Restrictions.

IMP-01: *Hardware Restriction* - A field programmable gate array (FPGA) will be used to buffer packets coming in and out of a secure end point. During this buffer, the FGPA will process all packets to determine if a packet is flagged for encryption/decryption. If flagged, the FGPA will encrypt/decrypt the packet and retransmit the packet.

IMP-02: *FPGA Software* – This project will require the use of a hardware description language (HDL) capable of configuring an FPGA.

IMP -03: *Custom Cryptographic Module* – The cryptographic module will be a custom design based on the request of the customer.

4 TED Architecture.

The FPGA platform being used is the Terasic DE2-115 board that utilizes a Cyclone IV FPGA chip from the manufacturer Altera. The architecture model of the DE-115 is shown in figure 4-1. The NIOS processor will be running our designed packet processing software that will perform deep packet inspection of every single packet that moves through the TED. The Encryption and Decryption DMA modules consist of a designed DMA and a designed cryptographic block cipher. The design of the DMA is meant to provide the cryptographic cipher with a way to interface with the packet processor in a way that will be much faster than with the CPU.

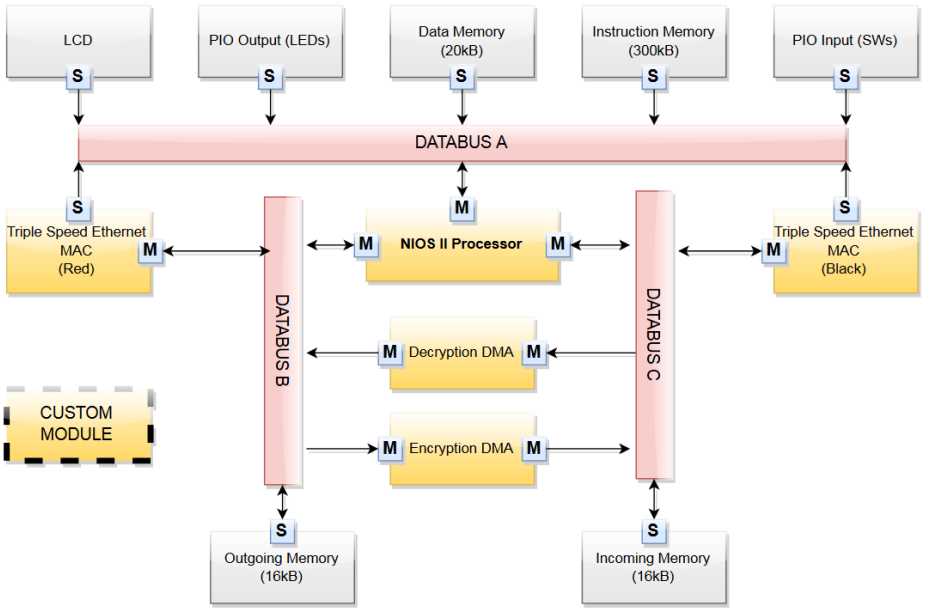


Figure 4-1: Architecture Model

The architecture of the TED will be realized within software. Integrated Development Environments (IDE) will be used to translate software code into hardware builds. The programs that we will employ are summarized below:

Quartus II software will be used to write all custom hardware interface modules using very high speed integrated circuit hardware description language (VHDL). These modules will be defined using the Avalon interface specification, which is an open-source interface specification used by the standard central processing unit (CPU) and peripherals included with the Altera suite.

QSYS is used to incorporate standard modules, such as on-chip FPGA random access memory (RAM)/read-only memory (ROM), a NIOS II standard processor, as well as any custom modules that we define for the TED. QSYS also creates a memory map for all I/O and peripherals, assignment of interrupt priorities, as well as defining a system interconnect bus (that takes care of the creation of chip select signals for various peripherals) allowing for ease of our design of the TED, as that is not included in our scope.

Finally, after our hardware design is compiled and downloaded to our target board, the NIOS II software build tools (SBT) will be used to program hardware drivers for our peripherals, DMA control algorithms, and deep packet

inspection and packet processing. The language used in the SBT is C, and the environment is a custom Eclipse distribution.

Figure 4-2 provides a summary of each program and the modules that will use them.

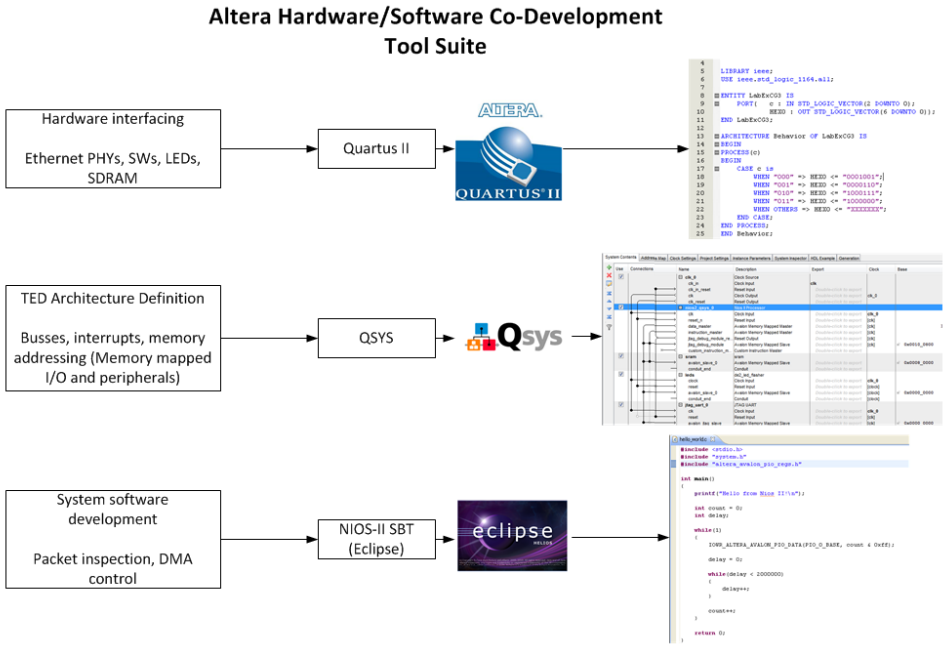


Figure 4-2: Altera Hardware/Software Tool Suite

Figure 4-3 shows our development flow as we progressed through the project. There were no major deviations on the execution of our development.

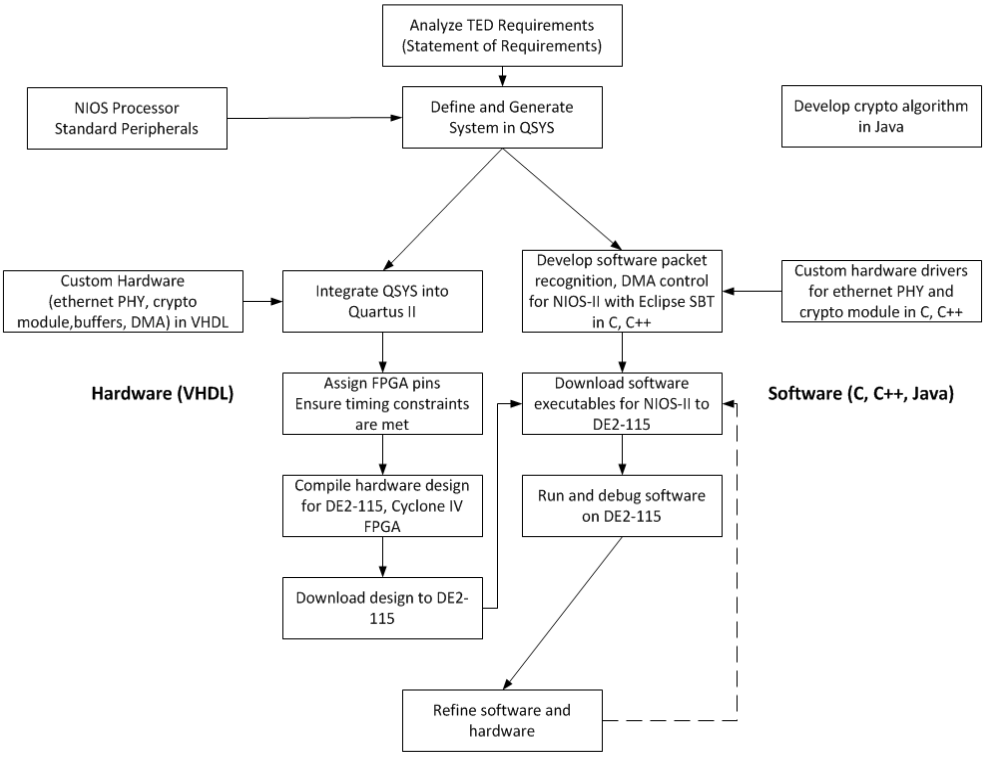


Figure 4-3: TED Development Flow

5 Detailed Design.

5.1 Overview.

The modules within the TED were designed and implemented using ANSI C and VHDL. With the exception of the DMA, all modules are portable to other chipsets that support standard VHDL and compile ANSI C.

5.2 Limitations.

5.2.1 NAT Operation.

The TED was tested and designed to operate on a static NAT configuration.

5.3 Modules.

5.3.1 Cryptographic Module.

5.3.1.1 Cryptographic Design Specification.

There are three specific requirements that our algorithm must meet, which are inherited requirements from the system level requirements:

The algorithm must not have a significant impact on system level latency or bandwidth.

The algorithm must be able to produce a different cryptographic hash for identical packets.

The algorithm must be able to produce a different cryptographic hash for identical blocks.

5.3.1.2 Cryptographic Design.

The first step for the algorithm was deriving the constants of the algorithm. A block size of 32-bits was chosen, simply due to alignment with our 32-bit processor. This meant that two 32 bit values could be XOR together in a single clock cycle. A key size of 56-bits was chosen. This would be made up of 3 parts:

1. 8-bit key: bits(55 downto 48)
2. 16-bit key: bits(47 downto 32)
3. 32-bit key: bits (31 downto 0)

The purpose of this key arrangement was to ensure that during key generation, an 8-bit pattern was left unmasked, which follows that each of the keys must not equal zero during generation.

The requirement that each packet must produce a different cryptographic hash is a very common issue with block ciphers. Figure 5-1 demonstrates visually the side effect of having the same hash function for each packet. The left image is the source, the middle image is with no salt, and the final image is the desired end state.

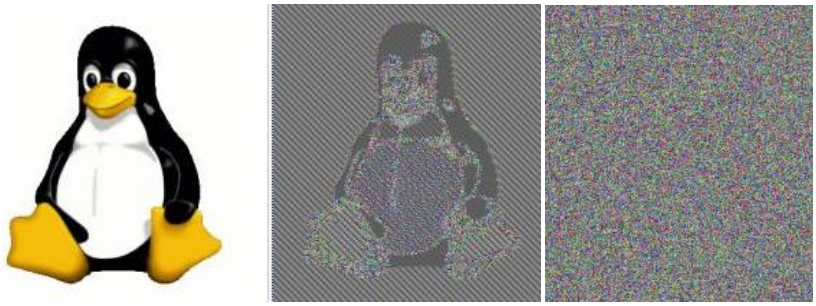


Figure 5-1: Demonstration of Weak Cryptographic Hash

The solution to this requirement is to introduce a unique value that would pair with the encryption key to produce a variation in the hash. This variation value is known as a salt. The salt that we used for each packet was derived from the identification value found within the layer IP header. This is a 16-bit value, so it was expanded with itself in order to provide a 32-bit salt. An identification value of `0xE74b` will result in a 32-bit salt of `0xE74bE74b`

The 32-bit salt would only be used to provide the first initialization vector (IV). Each subsequent block will be salted by the cryptographic hash of the preceding block, as shown in figure 5-2. Figure 5-3 shows the decryption model, notice that the major difference between the two methods is that the initialization vector is applied after the cipher block.

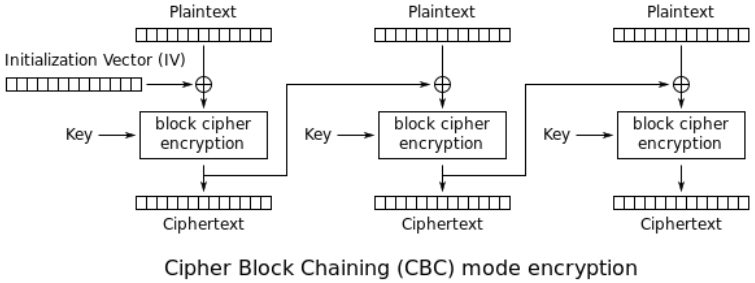


Figure 5-2: Encryption Structure for a Block Chain Cipher

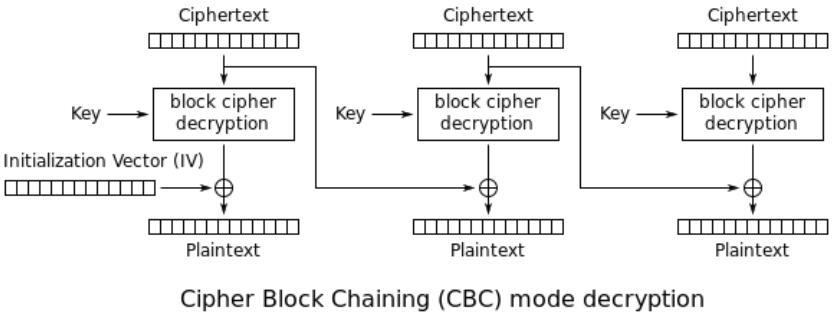


Figure 5-3: Decryption Structure for a Block Chain Cipher

The final requirement is to ensure that each block within a packet is hashed differently. Consider the case where there are more than 5 blocks of data, and each block contains an identical set of data. If the data is identical, the salt for any

block will be identical to the salt of the block that came before it. This greatly reduces the security of the cipher. The solution to this requirement was a key rotation algorithm to ensure that each block is encrypted with a different subset of the original key. This allows the data to remain the same, but produce a different salt for the next block. So for each block that is processed, the 8,16 and 32 bit keys is mutated to produce a different hash for each block of data.

5.3.1.3 Internal Structure.

Referring to Figure 5-3 and Figure 5-4 for encryption and decryption, our cryptographic design is placed within the cipher block. Figures A-1 and A-2 in Annex A below show the encryption and decryption models for how the algorithm is applied within the cipher block.

5.3.1.4 Operation.

The requirements of the cipher mentioned earlier in the paper outlined the importance of hashing and data repetition. The application of the cipher was on a per packet basis. This means that the first 32 bits of each packet will be block 0, and each packet would have a different salt value. To demonstrate the importance of this, some sample sets of data are shown to highlight the importance of these requirements.

For the following tables, the following sets of data were used.

- Plaintext Contents: aaaa aaaa aaaa aaaa aaaa (5 32-bit blocks of ASCII a)
- Ascii Hex Code for 'a': 0x61
- Salt 1: `0x47112015`
- Salt 2: `0x20154711`

Table 5-1 shows the result of the cipher with no salt and no rotation. The final hash of this packet is 988F3FF9. This hash was produced in this mode every single time it processed a packet with the same data. Also, the values after each block is processed cancel each other out. As the cipher text from a preceding block is passed to the next block, it reverses the operation performed on it. As a result, we are left with a hash that is only one operation away from decryption.

Table 5-1: Cipher Output with no Salt or Rotation

Salt	None
Value after the 1st operation:	`0x988F3FF9`
Value after the 2nd operation:	`0x0`
Value after the 3rd operation:	`0x988F3FF9`

Value after the 4th operation:	`0x0`
Value after the 5th operation:	`0x988F3FF9`

Table 5-2 shows the results of the cipher with different salts but the same data, and no rotation. The same issue of rotation where the operations cancel each other out exist, but this time we have the same data with different hashes. The salt value ensures that data is processed differently each time. This helps eliminate table lookup attacks on this cipher.

Table 5-2: Cipher Output with Salt

Salt:	`0x47112015`
Value after the 5th operation:	`0xDF9E1FEC`
Salt:	`0x20154711`
Value after the 5th operation:	`0xB89A78E8`

Table 5-3 shows the final iteration of our cipher development. The addition of a rotation algorithm on the key values is added. This rotation ensures that each block will be processed differently.

Table 5-3: Cipher Output with Rotation and Salt

Salt	0x47112015
Value after the 1st operation:	**0xDF9E1FEC**
Value after the 2nd operation:	**0xA689B540**
Value after the 3rd operation:	**0xA07D94C2**
Value after the 4th operation:	**0x37DD6ABF**
Value after the 5th operation:	**0x55A8AFBC**

The structure of the algorithm with 32, 16 and 8 bit keys requires that we do not produce a repeating pattern with the key throughout the encryption of the packet. The initial condition of the keys must not be reached for a valid packet size of 1500 bytes. The rotating mechanism that we have chosen allows us to rotate the 16-bit key sixteen times, the 8-bit key eight times, and each rotation of the 16-bit and 8-bit keys allows the use of 31 rotations of the 32-bit key. Formula 1 shows that the equation for the amount of blocks that can be processed with unique values of our rotating key set. The final value of this is 3255 blocks as shown in formula 5-1. As a comparison, a 1500 byte packet contains 375 blocks including the header data.

Formula 5-1: Block Repetition

$$Block\ Repetition = (31 * 15 * 7) = 3255\ Blocks$$

5.3.1.5 Error Handling.

The error handling for this module will rely on the higher layer network mechanisms for error detection such as TCP and IP protocols on the end user workstations. In the event of an error, the packet will simply be discarded and retransmitted.

5.3.2 CPU/Memory Module.

5.3.2.1 CPU/Memory Design Specification.

This CPU and Memory modules will be responsible for the packet decomposition and target selection. The packet decomposition must perform the following operations:

1. Determine the size of the incoming frame.
2. Determine the size of the present headers up to the transport layer
3. Determine the start point of the data

The target selection must be able to compare the incoming packet data with list of encryption and decryption targets it has stored in memory.

The packet processor receives a notification that the packet is ready for inspection from MAC controller. The value is passed to the packet processor as a void pointer that represents the start of the packet. Initially the packet shows up as a block of data, and the end state is to return whether or not to encrypt the packet, and which interface to send the packet on.

5.3.2.2 CPU/Memory Design.

The function of the packet processor will be a relative bit offset from the initial byte of the packet. Each packet will have constants that can be relied upon. Figures 5-4 shows the Ethernet and IPv4 pairing that is expected. The packet processor will be looking for this packet, and if it does not see it, it will forward the traffic without further processing. The dark fields denote a portion of the header that we use to perform the relative bit offset mechanic that allows us to deconstruct the packet.

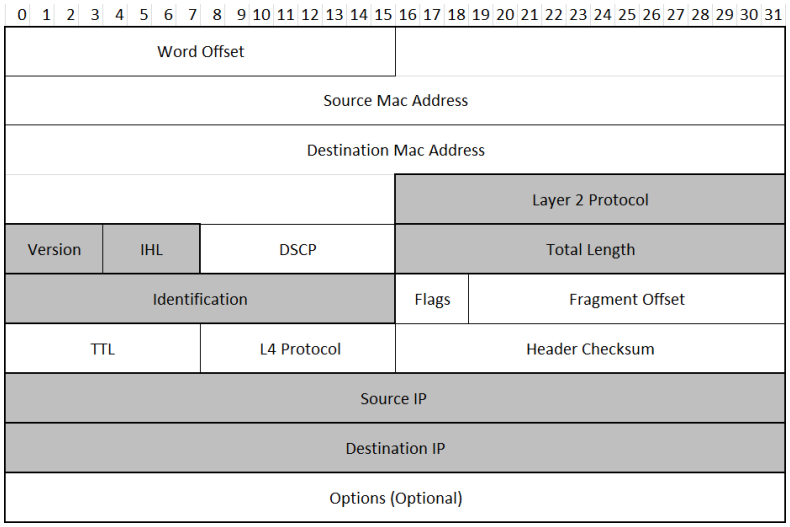


Figure 5-4: Ethernet and IPv4 Headers

Figure 5-5 and 5-6 show the only choice point we enter into with our program. Once an Ethernet and IPv4 packet are confirmed, the next layer expected is a TCP or UDP packet. If neither is found, the packet is forwarded without further processing.

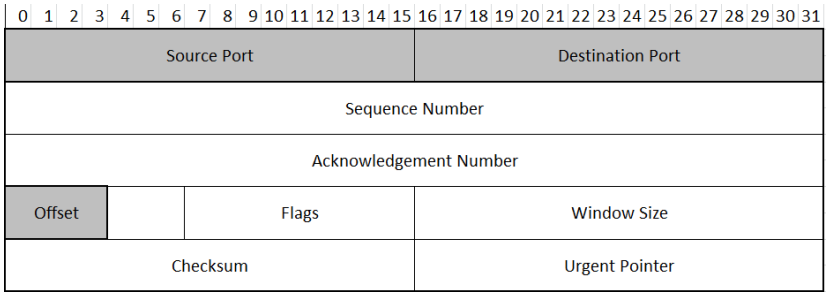


Figure 5-5: TCP Header

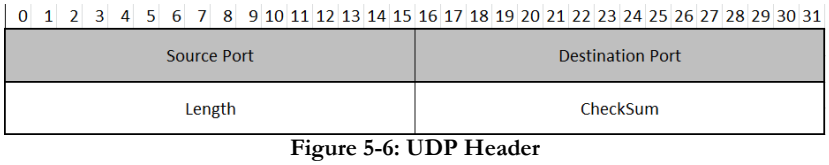


Figure 5-6: UDP Header

Figures 5-7 and 5-8 show a graphical view of the packet prior to entering the packet processor and the boundaries that are created once it leaves the packet processor respectively.

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0000
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0010
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0020
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0030
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0040
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0050
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0060
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0070
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0080
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0090
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0100
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0110
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0120
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0130

Figure 5-7: Packet Processor Initial View

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00	09	0f	e5	58	8e	30	85	a9	9b	a8	45	08	00	45	00	0000
05	dc	0b	6c	40	00	80	06	30	91	c0	a8	00	0e	17	17	0010
e1	51	47	23	01	bb	64	2f	a4	7c	75	a9	39	15	60	10	0020
00	fb	74	0b	00	00	77	77	77	77	x	x	x	x	x	x	0030
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0040
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0050
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0060
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0070
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0080
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0090
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0100
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0110
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0120
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	0130

Figure 5-8: Packet Parser Final View

5.3.2.3 Internal Structure.

Figure 5-9 shows the function relationships of the software. All the code inside NIOS II block is part of the packet processor written entirely with ANSI C. The relationships that move across the boundaries are VHDL functions.

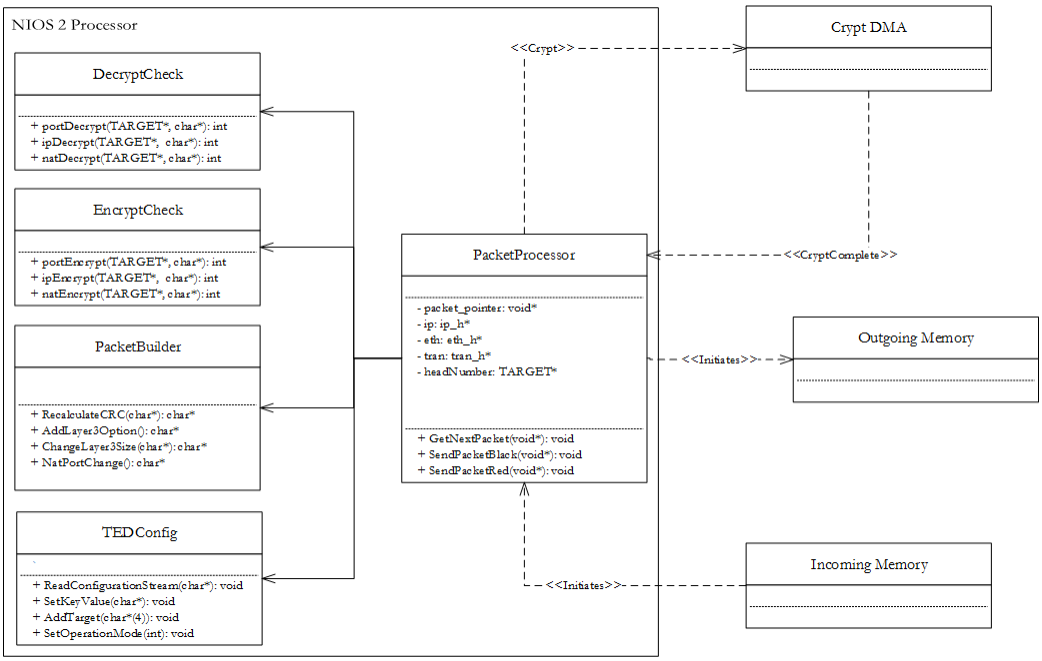


Figure 5-9: Packet Processor Structure

Figure 5-10 is the data structure of the packet processor. In order to dissect a packet, a series of buffers are taken relative to where the packet pointer is located in the code. ANSI C is not an object orientated language, however utilizing the struct type, it is possible to create some reusability within the code. Structs allow us to apply a rigid template to place over each packet, which make the code very stream lined and much more manageable than bit masking or void pointers. The other data type is the TARGET type, which is a linked-list. This allows the packet processor to perform lookups on the targets that will flag the TED when a packet requires encryption or decryption.

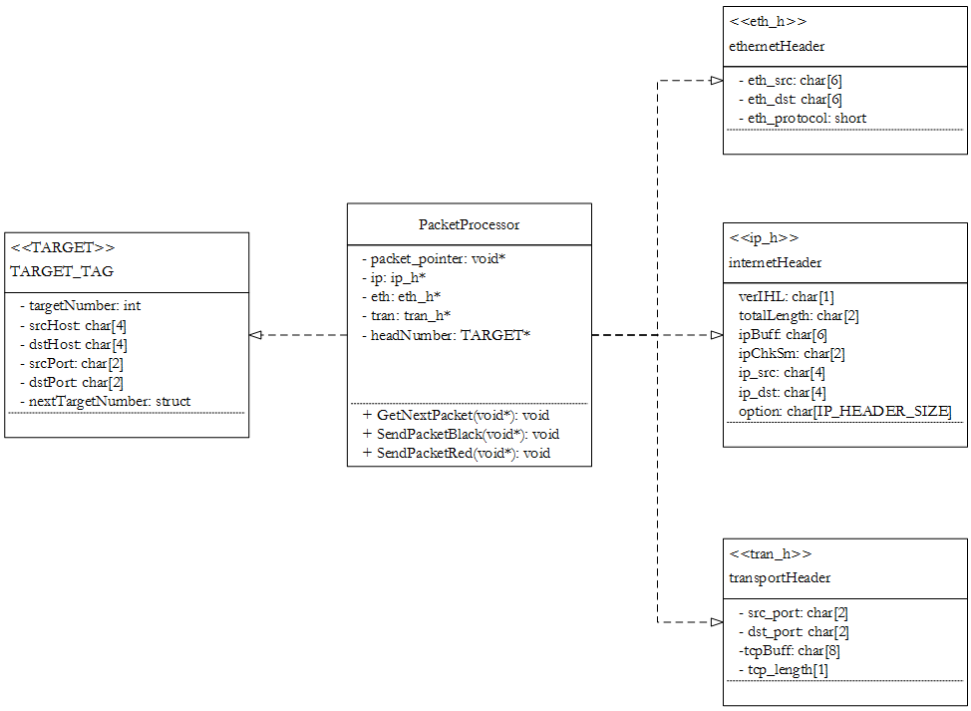


Figure 5-10: Packet Processor Data Structure

5.3.2.4 Operation.

Figure 5-11 shows the dynamic view of the lifetime of a packet as it transverses the packet processor .

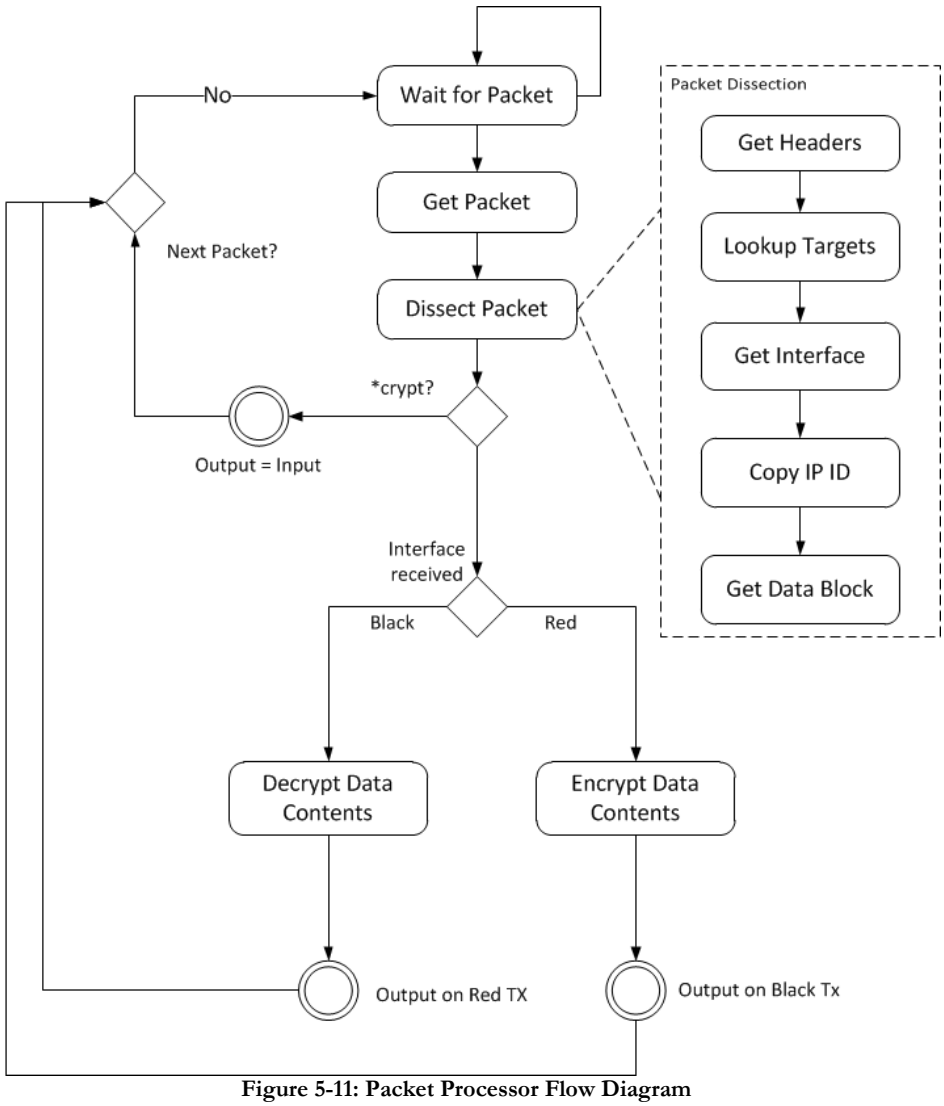


Figure 5-11: Packet Processor Flow Diagram

5.3.2.5 Error Handling.

The module relies on the end points for error detection and correction. If the TED causes and error it will rely on the station to request the entire packet as indicated by the TCP sequence numbers.

5.3.2.6 Miscellaneous.

The packet processor also handles administration functions for the TED. The TED will be looking for a very specific UDP packet, destined for IP address 10.10.10.10 with a payload that defines the key value and three four-tuples of trigger information: Source and destination IP address as well as source and destination port numbers.

The data format for the administration packet will have the following structure.

1. 56-bit Key Value
2. Target[1], Target[2], Target[3]

Figure 5-12 shows the allocation of the key and four-tuple data structure.

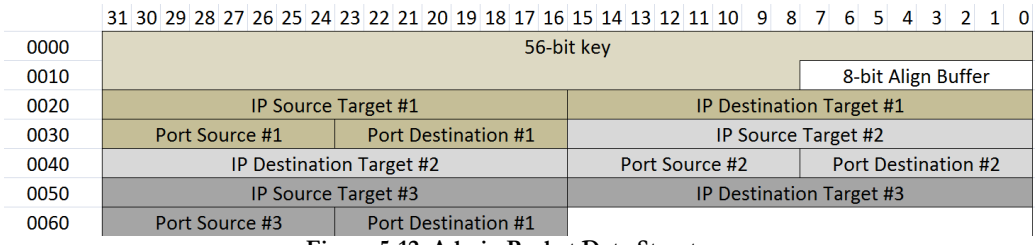


Figure 5-12: Admin Packet Data Structure

Received configurations will not be appended. Each configuration packet received will replace the old configuration in its entirety.

5.3.3 DMA Module.

5.3.3.1 DMA Design Specification.

The DMA modules inside the TED must be able to interface with the Avalon bus communication protocol. In addition they must be able to transfer data independently of the processor with a minimum of intervention, using either memory mapped semaphores or interrupts to signal they are finished with transfers.

The cryptographic module will also use a form of DMA in order to complete the encryption and decryption operations.

5.3.3.2 DMA Design.

When looking at Figure 5-13, the interfaces in the architecture that required the use of DMA, and their types are:

1. Triple speed Ethernet MAC Red Receive (Red Rx) interface. This required the use of a streaming-to-memory-mapped (MM) DMA in order to write incoming streaming data as it is received from the red receive interface, to sequential locations in the outgoing memory block.
2. Triple speed Ethernet MAC Black Transmit (Black Tx) interface. This required the use of a MM-to-streaming DMA in order to write data from sequential locations in the outgoing memory block, to a stream to the black transmit interface.
3. Triple speed Ethernet MAC Black Receive (Black Rx) interface. This required the use of a streaming-to-MM DMA in order to write incoming streaming data as it is received from the black receive interface, to sequential locations in the incoming memory block.
4. Triple speed Ethernet MAC Red Transmit (Red Tx) interface. This required the use of a MM-to-streaming DMA in order to write data from sequential locations in the incoming memory block, to a stream to the red transmit interface.
5. Encryption. This required the use of a MM-to-MM DMA in order to read-encrypt-and-write, sequential data located in the outgoing memory block.

6. Decryption. This required the use of a MM-to-MM DMA in order to read-encrypt-and-write, sequential data located in the incoming memory block.

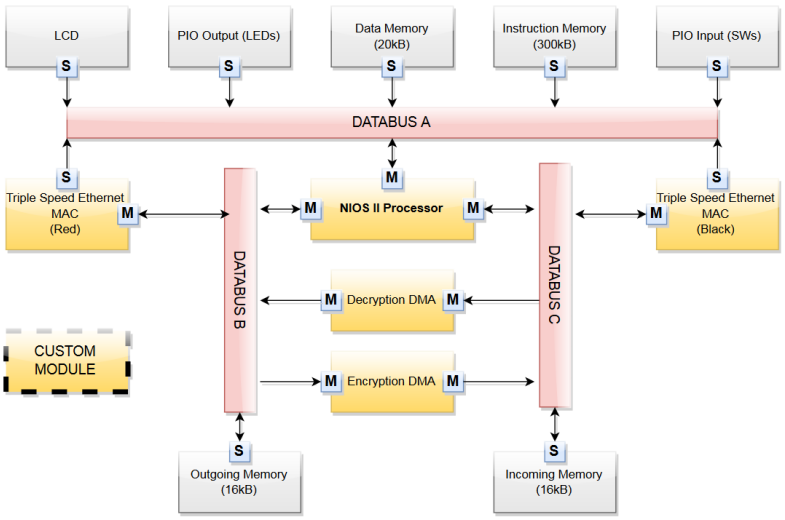


Figure 5-13: DMA members

5.3.3.3 Internal Structure.

Figure 5-13 shows the life of a packet as it is received on the black receive interface. Specifically, our design uses the streaming-to-MM DMA in order to move the data to one of sixteen, 2048 byte blocks of memory that are allotted by a linked list structure. While it is stored in the linked list structure, the packet processor software function will inspect the packet, and determine if it is coming from a paired location protected by another TED and must be decrypted, or a packet that is simply actively ignored. Once the packet processor makes the determination of a decryptable packet, it will trigger the decryption MM-MM DMA, and then the red transmit MM-to-streaming DMA. If the packet is to be actively ignored, it is simply sent to the red transmit interface by its MM-to-streaming DMA.

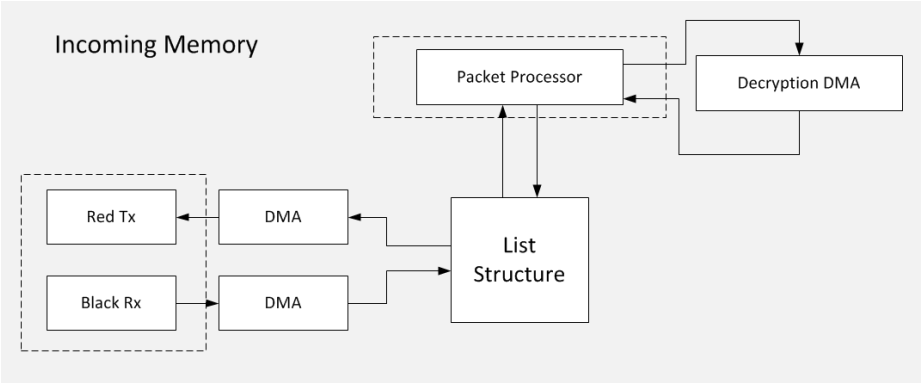


Figure 5-14: Typical DMA Transfer of Control (Incoming)

Figure 5-14 shows the life of a packet as it is received on the red receive interface. The design uses the streaming-to-MM DMA in order to move the data to one of sixteen, 2048 byte blocks of memory that are allotted by a linked list structure. While it is stored in the linked list structure, the packet processor software function will inspect the packet, and determine if it is and administration packet from the administration graphical user interface (GUI), coming from a

paired location protected by another TED and must be decrypted, or a packet that is simply actively ignored. Once the packet processor makes the determination of a decryptable packet, it will trigger the decryption MM-MM DMA, and then the red transmit MM-to-streaming DMA. If the packet is to be actively ignored, it is sent to the red transmit interface by its MM-to-streaming DMA. If the packet is an administration packet, the packet processor will use deep packet inspection to read the configuration data, set its registers appropriately, and then return the block of memory to the pool available for packet reception on the red receive interface.

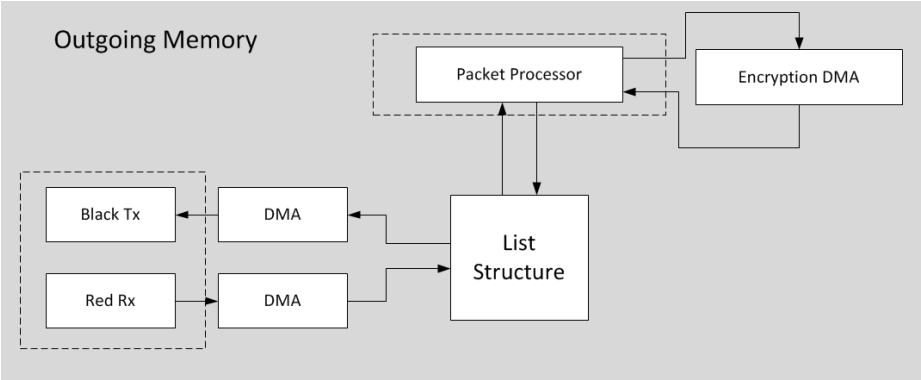


Figure 5-15: Typical DMA Transfer of Control (Outgoing)

Figure 5-15 shows the linked list structure that controls the organization and allotment of the sixteen 2048-byte packet buffers in both the incoming and outgoing memory blocks. Nodes on the below linked lists that maintain packet information and pointers to the specific bufferare moved from list to list when buffers are: allocated for frame reception, received a frame, inspected, encrypted/decrypted, prepared for transmission, transmitted, and then made available again.

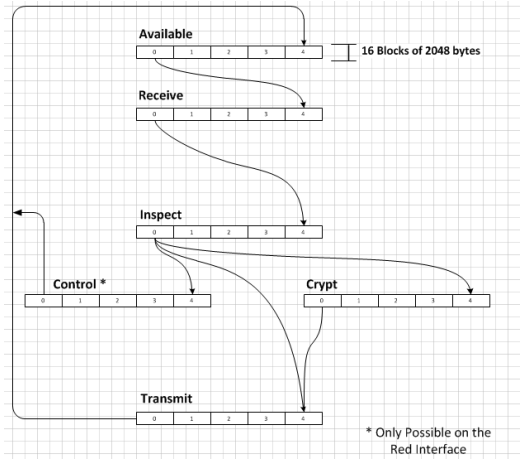


Figure 5-16: Packet Buffer Linked Lists

5.3.3.4 Operation.

Several approaches were assessed in order to control the operation of the DMAs and flow of packets in two directions at once. Initially, attempts were made to use a zero-size main loop, and rely on interrupts to trigger specific code blocks in the event of frame reception, encryption, and frame transmission, etc. Early design efforts failed in that due to the sheer number of interrupts generated, the processor was overwhelmed with the amount of context-switching that must take place on each interrupt, and was unable to operate in a meaningful way. The ultimate design uses a super-loop in the

main function that polls each linked list in turn, performs specific operations on that linked list buffer, and then moves the node to another linked list in order that it is acted on appropriately by that list. In turn, each of the linked lists' operations and node transitions is as follows.

1. The available linked list is inspected. All nodes (which point to a specific 2048-byte block) are moved to the receive linked list.
2. The receive linked list is inspected. All nodes that have received packets are moved to the inspect linked list.
3. The inspect link list is inspected. All nodes are run through the packet processor which determines if the packet is an administration packet, if it is supposed to be encrypted/decrypted (based on deep packet inspection if it is communication between two paired TEDs), or whether it is to be actively ignored. It is then moved to the control linked list, the crypt linked list, or the transmit linked list respectively.
4. The control linked list is inspected. All nodes that contain configuration information/cryptographic keys, are loaded onto the TED. The nodes is then moved to the available linked list.
5. The crypt linked list is inspected. The layer four payload data of all nodes are encrypted/decrypted (depend which interface the packet was received from). The node is then moved to the transmit linked list for transmission.
6. The transmit linked list is inspected. All nodes that have been transmitted are moved to the available linked list.

It is important to note that these lists are duplicated for each of the outgoing memory and incoming memory, in order that packets are moved in different directions through the TED simultaneously, achieving duplexivity through the TED.

5.3.3.5 Error Handling.

Error handling of all DMAs is permitted to trickle down to the transmission of packets through the opposing interface. The IEEE 802.3 standard provides for robust handling of errors during transmission through the use of checksums in different layers of a packet. This robust handling of errors does not interfere with operation of the TED and allows it to degrade gracefully and remain operational if errors happen during packet inspection or encryption. In short, if a packet fails checksum verification during any link of transmission it is simply dropped and it is left to the individual protocol on the secure endpoints to proceed.

5.3.4 Parallel I/O Module.

5.3.4.1 Parallel I/O Design Specification.

The parallel I/O was required to assist in development. The various mechanisms available on the DE-115 board allowed us to troubleshoot outside of the software. This reduced the amount of configuration needed within the code and also provided instantaneous feedback in terms of the interface states.

Specific functions and register settings were bound to switches, LEDs and displays:

1. PHY Chip Settings:
    a. Control link speed (10/100/1000/Auto).
    b. Control duplexity.
    c. Crossover enabled or disabled
    d. Alert when a packet is received or transmitted on any interface.
2. Enable Cryptographic module
3. Cryptographic filtering modes
4. Reset the TED

5.3.4.2 Parallel I/O Design.

The design elements of this module were connecting the appropriate signals to the IO elements that will display the data. We also didn't want to slow down the primary operations of the TED, so the operations were placed on a completely separate bus. To further separate the primary operations of the TED, we are using an asynchronous reset button to drive any setting changes. The reset button also acts as a way to test a multitude of settings rapidly from the TEDs initial state.

The code development behind each switch and LED varies considerably, and can be referenced in the source code.

5.3.4.3 Internal Structure.

Each I/O element is a memory mapped interface to the CPU as shown in figures 5-11, 5-12, and 5-13. On initialization the TED will cycle through each switch and appropriately trigger each of the settings. To make initialization occur more rapidly, a reset switch was used to force the board to re-initialize. The reasons for this were two-fold. The first is to have a quick way to power cycle the board without removing the power, and the second was to remove the requirement to loop through and constantly check for any I/O changes.

5.4 Interface Descriptions.

Following in this section is a brief description of all physical interfaces we used on the DE2 115 development board, as detailed in figure 5-10.

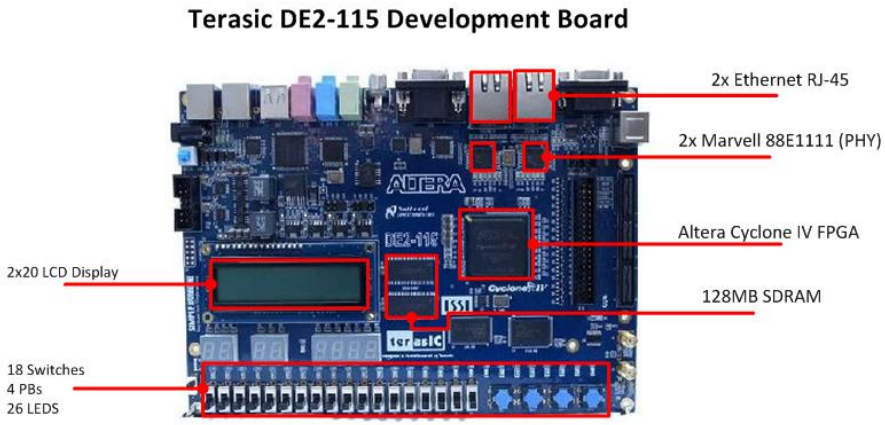


Figure 5-17: Terasic DE2-115 Development Board Interfaces

5.4.1 All FPGA Internal Interfaces.

All internal FPGA interfaces will be of the signal type "STD_LOGIC" or "STD_LOGIC_VECTOR (# DOWNTO #)" specified by the IEEE VHDL libraries included in the Quartus II software, and they will conform to the Avalon open source interface specification that is primarily used by Altera. This interface specification allows the Qsys software to map and create the proper module interconnection logic and relieve the overhead from the programming team.

5.4.2 Ethernet.

The external Ethernet interfaces in figure 4-7 are identical and are copper Ethernet RJ-45 connections communicating over twisted pair CAT-5 wiring. The internal interface between the FPGA and Marvell 88e1111 PHY are of the type "STD_LOGIC" or "STD_LOGIC_VECTOR (3 DOWNTO 0)", as well as an on board oscillator used by the PHY.

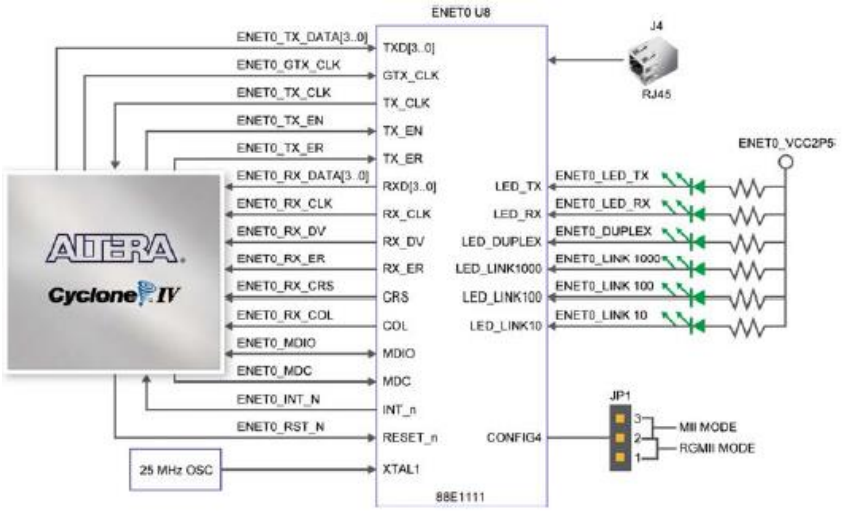


Figure 5-18: Ethernet Interfaces

5.4.3 LCD.

The external LCD interface will be controlled by FPGA signals of the type "STD_LOGIC" and are wired directly to certain pins on the FPGA as described in figure 4-8.

Figure 5-19: LCD Interface

5.4.4 Parallel I/O.

The parallel interface has two function controlled by the memory-mapped interface to the CPU. It reads information from switches and displays discrete information on LEDs. All signals are of the type "STD_LOGIC", whether they are to or from the FPGA as shown in figures 4-9, and 4-10.

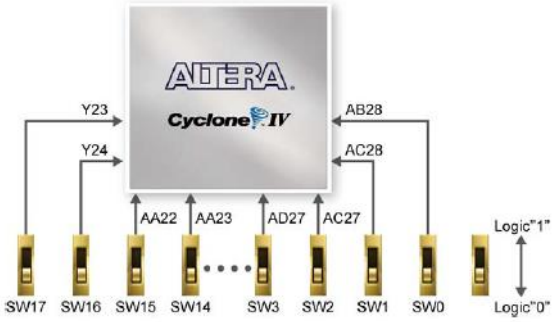


Figure 5-20: DE-115 Switches Interface

Figure 2-21: DE-115 LEDs Interface

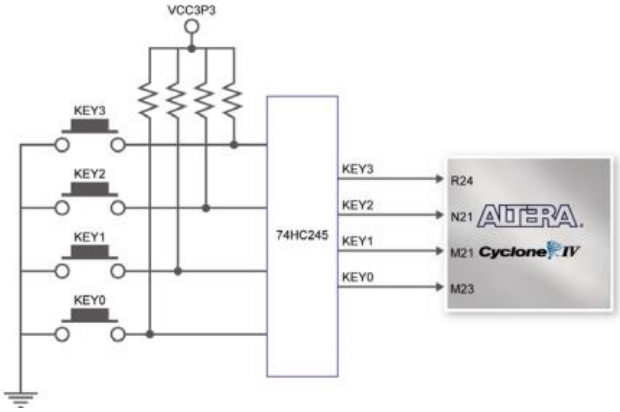


Figure 5-22: DE-115 Switches Interface

5.4.5 SDRAM.

The SDRAM interface in figure 4-10 provides access to 128MB of external RAM in the form of two external 64MB chips that are wired to a 32bit high-low half word configuration. All signals communicating to the SDRAM are of the type STD_LOGIC" or "STD_LOGIC_VECTOR (# DOWNTO #)".

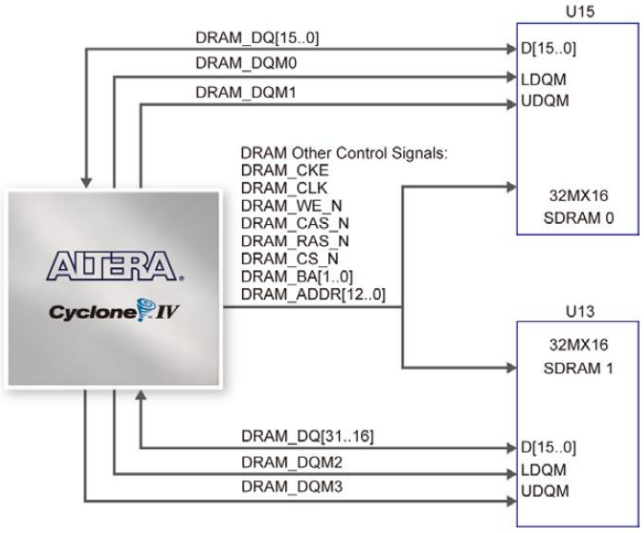


Figure 5-23: SDRAM Interface

6 Equipment.

The equipment that was used for testing and development is categorized in Table 6-1.

Table 6-1: Final Project Equipment List

Quant.	Name	Function
2	**Terasic DE2-115 Development Board**	**TED**
2	**Raspberry Pi Model B**	**Red Server (User 3) & Black Sensor**
2	**Cisco 2960G Catalyst Switch**	**Network Distribution**
1	**Cisco 2600 Router**	**Network Routing**

7 Results.

7.1 Testing.

7.1.1 Bandwidth Test.

To measure the bandwidth of the TED and demonstrate PR-01 (100Mbps throughput) we used the SCP protocol in verbose mode. This method would have consistent overheads for both configurations and provide a detailed report once the file was transferred. The smaller file sizes showed no significant difference in throughput, however as the file size increased, the throughput settled at approximately 20 Mbps. We believe this is due to an interrupt running on the FGPA. The FPGA must be connected via USB at all times to the Quartus software. This is a restriction enforced with the particular license we are using. Since we are showing no lag times in latency and smaller packet transfers, we believe this assumption is well founded.

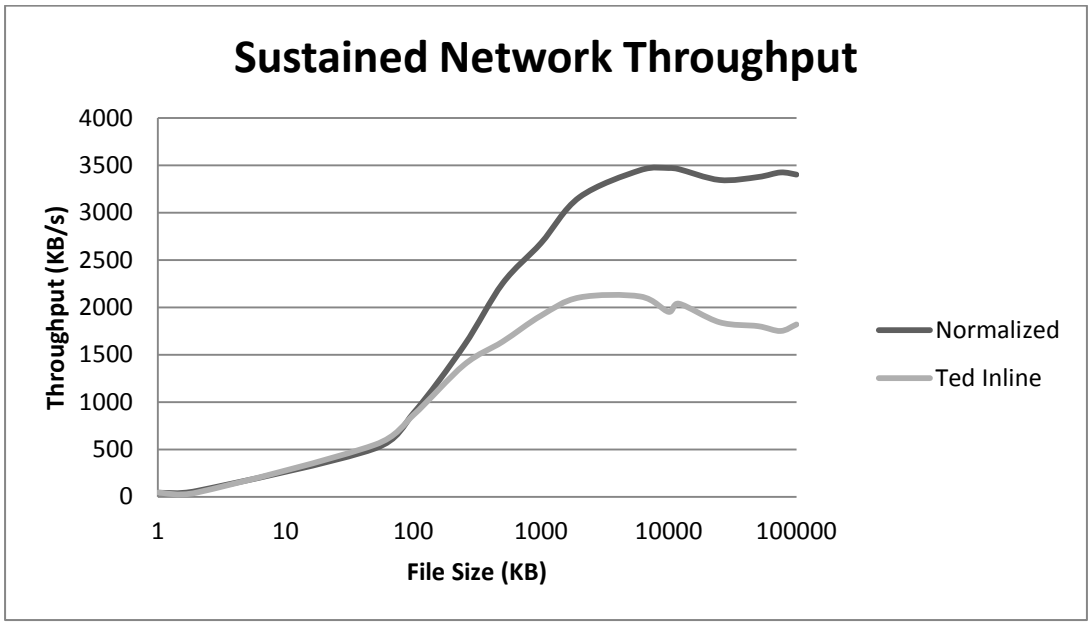


Figure 7-1: Comparison of Bandwidth with and without TED

7.1.2 Latency Test.

The latency was measured using the ping command. Sample sizes were between 200 and 300 pings with the average ping time shown in figure 7-2. Even the largest packet size only delays the latency by 1ms, well below the timing requirement of PR-02 of a 10ms delay traversing through the ted.

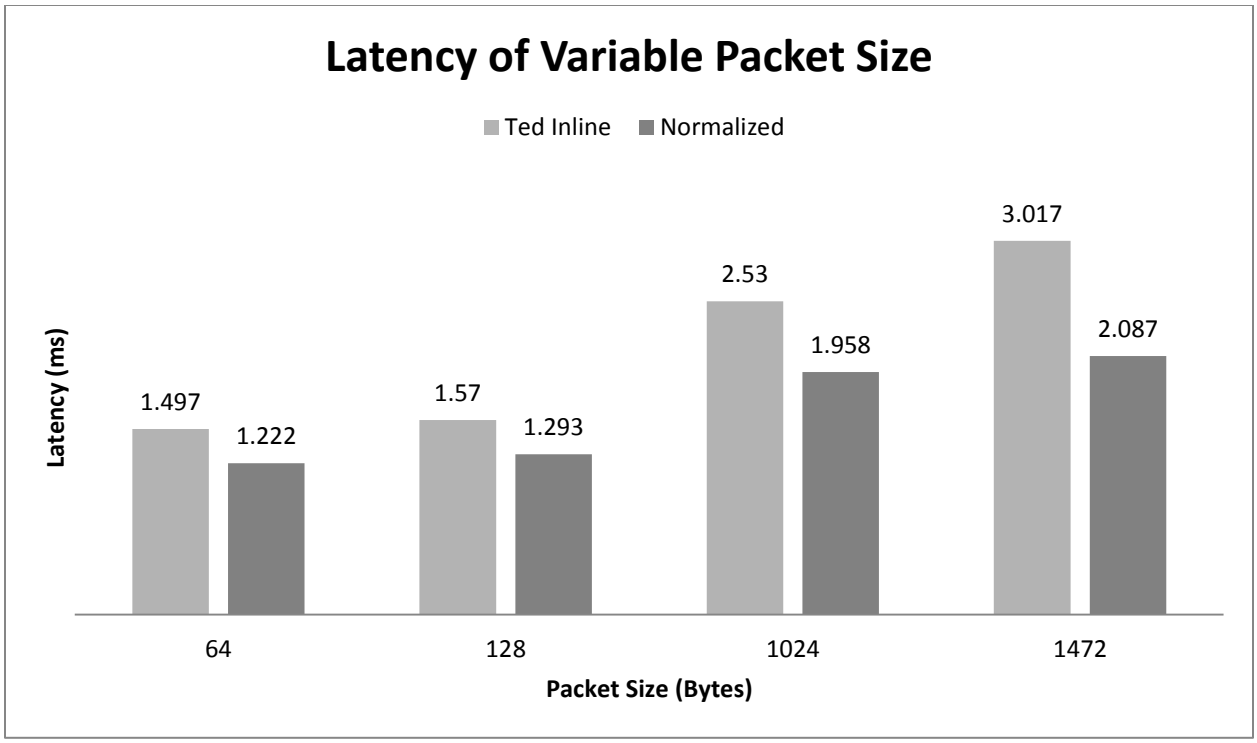


Figure 7-2: Latency of Variable Packet Size

7.1.3 Duplexity Test.

As part of IEEE 802.3 standard, auto-negotiation of the end stations are successfully connecting with a full-duplex connection at 100 Mbps satisfying IR-01 (Duplexity).

7.1.4 NAT Test.

Requirement IR-02 (NAT Transversal) outlined that the TED must survive static NAT transversal. The NAT function of the TED was built into the target design. This means that NAT configuration on the TED is a function of how the targets are applied to each packet. When NAT is enabled, the trigger for the decrypt is now a result of the source and destination IP addresses. If a packet is received on a TED, and the source IP address was the public IP address of the sending TED, and the destination is for the private IP address. The major difference between NAT operation and normal operation is to know that static NAT is being used, and to know the public IP addresses of each network.

7.1.5 Administration Panel Test.

Requirement IR-03 Administration Interface outlined a secure user that was able to change the key and target settings of the TED. Section 5.3.3.6 outlined the structure of how the TED would process this data. It would require the key, followed by the 4-tuple of the source and destination IP/port pairings. This configuration was completed as a java swing GUI. The information was sent to the TED as a UDP datagram, with the payload being a character buffer of all the configuration changes being sent to the device. Figure 7-2 shows the GUI configuration and figure 7-3 shows the resultant

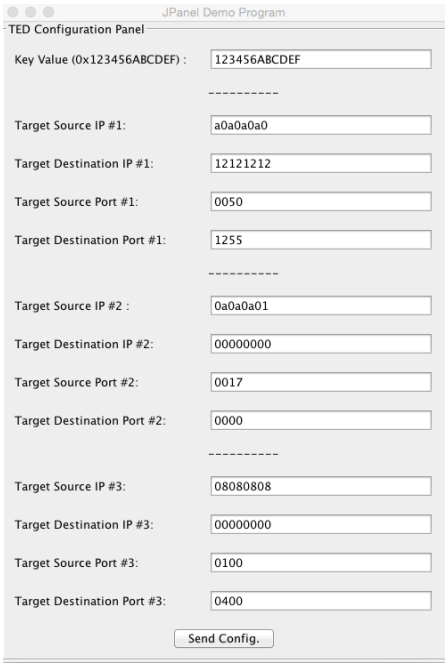


Figure 3-3: TED Administration GUI

```
SS:~ Count$ sudo tcpdump -X -i en3 host 10.10.10.10
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en3, link-type EN10MB (Ethernet), capture size 65535 bytes
20:45:38.338386 IP 192.168.1.5.61076 > 10.10.10.10.300: UDP, length 84
        0x0000:  4500 0070 72b4 0000 4011 3208 c0a8 0105  E..pr...@.2.....
        0x0010:  0a0a 0a0a ee94 012c 005c 3ab4 3132 3334  .......,.\:.1234
        0x0020:  3536 4142 4344 4546 6130 6130 6130 6130  56ABCDEFa0a0a0a0
        0x0030:  3132 3132 3132 3132 3030 3530 3132 3535  1212121200501255
        0x0040:  3061 3061 3061 3031 3030 3030 3030 3030  0a0a0a0100000000
        0x0050:  3030 3137 3030 3030 3038 3038 3038 3038  0017000008080808
        0x0060:  3030 3030 3030 3030 3031 3030 3034 3030  0000000001000400
```

Figure 7-4: TED UDP Administration Packet

8 Summary.

The purpose of this project was to develop a hardware based symmetric encryptor that would abstract the security of network data away from the operating system towards a dedicated device. Security software that runs on an operating system, inherits the security vulnerabilities of all other programs running on that operating system. To remove that attack vector will greatly increase the overall security of the system.

We placed the device inline of network communications and in doing so we can intercept data and modify the contents without obstructing the flow of network communications. This means, we do not have to transverse the protocol stack in order to process the data. This allows our device to encrypt data without leaving a signature or changing the header data in the packet. This also makes our device a layer two device, but it is fully aware of the entire contents of the packet.

This paper introduces 3 common use cases. These are the three use cases that we used to validate our design; however, there is no restriction to how this device may be used. It can operate WAN to WAN and even function as a data diode. These use cases are beyond the scope of this paper, but demonstrate that many possibilities for deployment exist; bound only to the interpretation of the user.

9 Conclusion.

The final state of the project demonstrates that the DE-115 FGPA development board is capable of conducting secure network communications over Ethernet with a custom cryptographic module. In addition, when the network was properly configured to operate in a NAT environment, the TED was able to survive two instances of NAT transversal.

Through a comparison of network operations with and without the TED placed inline network communications we have shown that there are negligible performance losses when using the TED. The final results of our testing showed a 0.5% decrease in bandwidth, and less than 0.1% decrease in latency, approximately 1/100 millisecond. The demonstration of duplexity was demonstrated by conducting synchronous communication between User 1 and User 3 and observing that no blocking waits were being initiated within the Quartus console.

We have established the security against some of the more vulnerable aspects to block ciphers such as rainbow tables, however since the data is wrapped in 32-bit blocks leaving the cipher open to brute force techniques. While the cipher successfully obscures the data, without a proper peer review it is impossible to verify that it is a secured algorithm. Its implementation must be viewed as an academic pursuit.

10 Discussion.

10.1 PR-04 Requirement Removal.

One major change we made from our preliminary design specification was the removal of performance requirement 04: The TED will insert a custom layer 4 header to mask any usage signatures from the device. This requirement was removed because we felt it conflicted too much with the idea of a transparent network device. We were trading a usage characteristic (ie: Traffic over port 23 can be characterized as telnet traffic) for an absolute signature. For this reason, we only encrypt the entire transport layer payload, and leave the transport layer as is.

10.2 Major Challenge and Lessons Learned.

Hardware implementation was the major challenge of this project. There were specific ways that the board allowed us to interface with the data. We required precise control over the bits that were entering the FPGA. We used a relative bit offset mechanism to perform deep packet inspection, and knowing the exact address of each bit was crucial. This piece alone accounted for the majority of all the work hours on this project. The lesson learned from this challenge was to give more respect to the proprietary software that is packaged with FPGA boards. We invested many hours to determine the raw specs of the FGPA that would meet our design requirements; instead we should have researched the tools and software that fit our project.

11 Future Work.

11.1 Cryptographic Cipher Iterations.

Additional iterations of our designed cipher are needed. There is a currently weakness in our implementation that can exploit the fact that the blocks are processed in the same way each time. If data is known to occur in the first 4 bytes of the packet, the key and salt pairing can be easily extracted against a brute force attack.

11.2 Packet Processing Depth.

The packet processor is programmed to only consider Ethernet, IPv4, TCP and UDP. Additional functionality can be built into the board to include IPv6, ICMP, ARP, VLAN and any other desired protocol.

Annexes

Annex A – Additional Diagrams and Figures

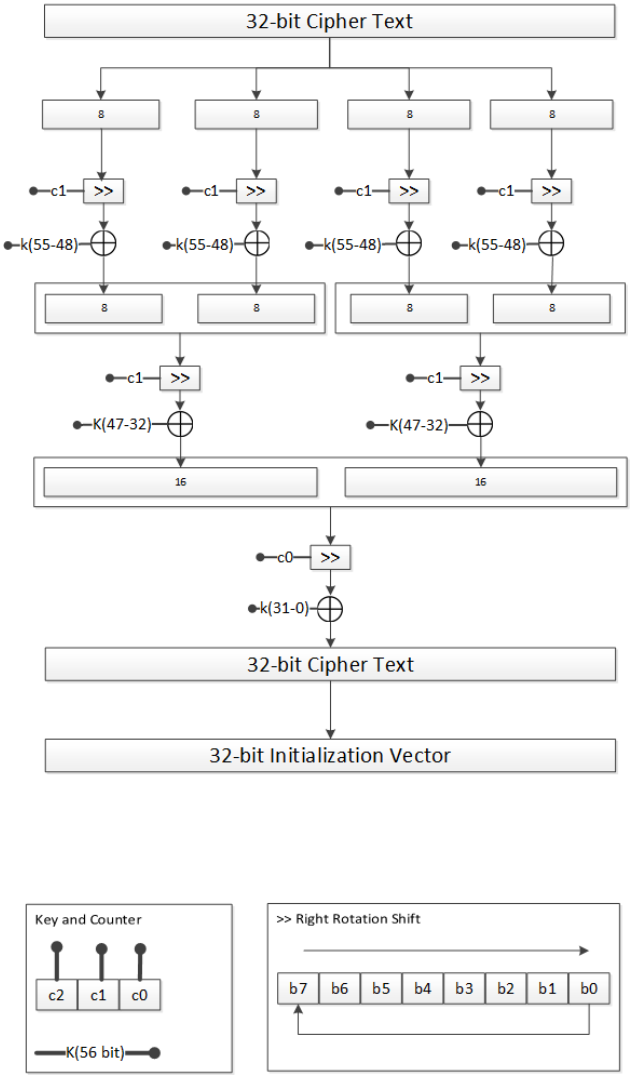


Figure A-1: Block Cipher Decryption Model

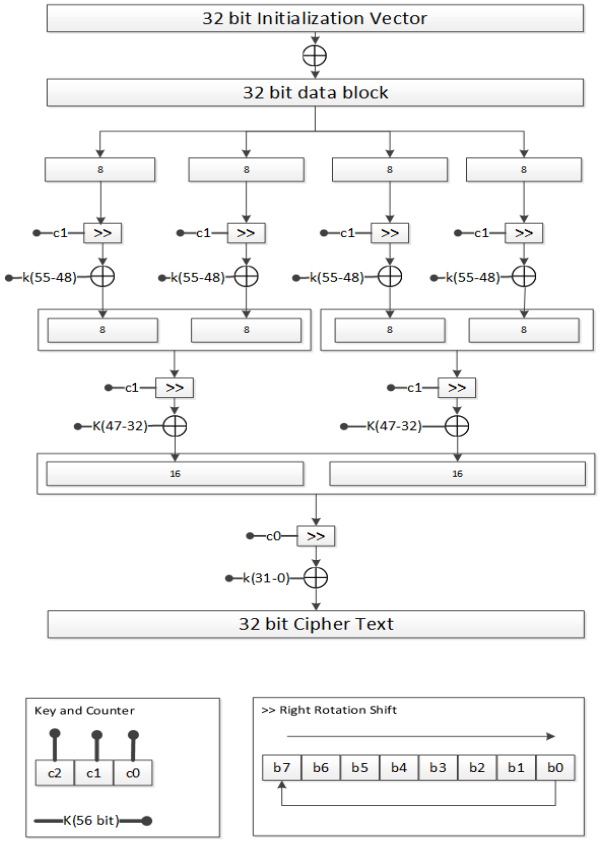


Figure A-2: Block Cipher Encryption Model